Data Structure

(Java programming)

Chapter 9.

- ***Ex1-Maps***

• Ex1-Maps

Maps?

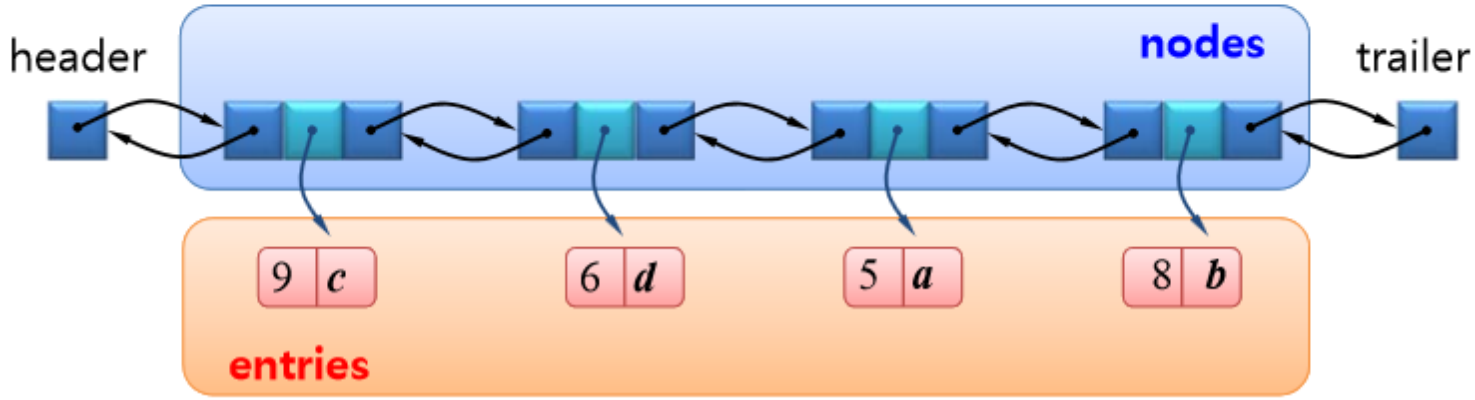
- A ***map*** models a *searchable* collection of (key, value) entries
  - The main operations of a map are for *searching*, *inserting*, and *deleting* entries based on their keys
  - Multiple entries with the same key are **NOT** allowed → each key must be unique
  - Sometimes referred as ***associative store*** because the key associated with a value determines its address or location

- Application
  - Student-record database (key = student ID)
  - Symbol table of compiler (key = name of a variable)
  - Web browser cache (key = URL)

• Ex1-Maps

A Simple List-Based Map

- We can efficiently implement a map using an *unsorted* list
  - We can store items of a map in a node list S in an arbitrary order (e.g., based on a doubly-linked list)



• Ex1-Maps

Map ADT operations

- `integer` **`size`**`(), boolean` **`isEmpty`**`()`
- `object` **`get`**`(`***k***`)`
  - If the map ***M*** has an entry with key ***k***, return its associated value; else, return null
- `object` **`put`**`(`***k***`, `***v***`)`
  - Insert entry (***k***, ***v***) into the map ***M***; if key ***k*** is not already in ***M***, then return null; else, return and replace with ***v*** the old value
- `object` **`remove`**`(`***k***`)`
  - If the map ***M*** has an entry with key ***k***, remove it from ***M*** and return its associated value; else, return null
- `list` **`keys`**`()`
  - Return a list of the keys in ***M***
- `list` **`values`**`()`
  - Return a list of the values in ***M***
- `list` **`entries`**`()`
  - Return a list of the (***k***, ***v***) entries in ***M***

❖ keys().iterator() returns an iterator of keys, so do values() and entries()

• Ex1-Maps

Searching Algorithms

```
Algorithm get(k):
    Input: A key k
    Output: The value for key k in M, or null if there is no key k in M
    for each node p in S.nodes() do
        if p.element().getKey() = k then
            return p.element().getValue()
    return null          {there is no entry with key equal to k}
```

❖ A ***linear search*** algorithm that checks each entry stored in the list ***S*** whether it has the given key ***k***

• Ex1-Maps

Insertion Algorithms

Algorithm put(k, v):
  Input: A key-value pair (k, v)
  Output: The old value associated with key k in M, or **null** if k is new
  for each node p in S.nodes() **do**
    if p.element().getKey() $= k$ **then**
      $t \leftarrow p$.element().getValue()
      S.set$(p, (k, v))$
      return t {return the old value}
  S.addLast$((k, v))$
  $n \leftarrow n + 1$ {increment variable storing number of entries}
  return null {there was no previous entry with key equal to k}

Check if any entry in the list ***S*** already has the given key ***k***

• Ex1-Maps

Deletion Algorithms

Algorithm remove(k):
  Input: A key k
  Output: The (removed) value for key k in M, or **null** if k is not in M
  for each node p in S.nodes() **do**
    if p.element().getKey() $= k$ **then**
      $t \leftarrow p$.element().getValue()
      S.remove(p)
      $n \leftarrow n - 1$ {decrement variable storing number of entries}
      return t {return the removed value}
  return null {there is no entry with key equal to k}

Search the list ***S*** for the entry with the given key ***k*** (i.e., entry to be removed)

• Ex1-Maps

Example

Operation	*Output*	*Map*
isEmpty()	**true**	Ø
put(5,*A*)	**null**	(5,*A*)
put(7,*B*)	**null**	(5,*A*), (7,*B*)
put(2,*C*)	**null**	(5,*A*), (7,*B*), (2,*C*)
put(8,*D*)	**null**	(5,*A*), (7,*B*), (2,*C*), (8,*D*)
put(2,*E*)	***C*** (old value)	(5,*A*), (7,*B*), **(2,*E*)**, (8,*D*)
get(7)	*B*	(5,*A*), (7,*B*), (2,*E*), (8,*D*)
get(4)	**null**	(5,*A*), (7,*B*), (2,*E*), (8,*D*)
get(2)	*E*	(5,*A*), (7,*B*), (2,*E*), (8,*D*)
size()	4	(5,*A*), (7,*B*), (2,*E*), (8,*D*)
remove(5)	*A*	(7,*B*), (2,*E*), (8,*D*)
remove(2)	*E*	(7,*B*), (8,*D*)
get(2)	**null**	(7,*B*), (8,*D*)
isEmpty()	**false**	(7,*B*), (8,*D*)

Map에 저장된 데이터들을 볼 수 있게 출력해야 함.
Ex) true null
null (5,A)
…

- ***Ex2-Map & Sorting***

• Ex2-Map & Sorting

Example

Operation	*Output*	*Map*
isEmpty()	**true**	Ø
put(5,*A*)	**null**	(5,*A*)
put(7,*B*)	**null**	(5,*A*), (7,*B*)
put(2,*C*)	**null**	(5,*A*), (7,*B*), (2,*C*)
put(8,*D*)	**null**	(5,*A*), (7,*B*), (2,*C*), (8,*D*)
put(2,*E*)	***C*** (old value)	(5,*A*), (7,*B*), **(2,*E*)**, (8,*D*)

You should sort the Map(List)

Results

= (2,E), (5,A), (7,B), (8,D)